Exhibit A

EXHIBIT A – Friedl

Invalidity Contentions for U.S. Patent No. 6,842,796
Based on *Mastering Regular Expressions: Powerful Techniques or Perl and other Tools*, Jeffrey E. F. Friedl, O'Reilly (1st Ed. 7th Printing, Dec. 1998) ("Friedl")

Based upon Plaintiff's Complaint, Infringement Contentions, and apparent claim constructions and application of the claims to Defendants' accused products, as best as they can be deciphered, the reference charted below anticipates or at least renders obvious the asserted claims. These invalidity contentions are not an admission by Defendants that the accused products are covered by or infringe the asserted claims, particularly when these claims are properly construed and applied. These invalidity contentions are not an admission that Defendants concede or acquiesce to any claim construction implied or suggested by Plaintiff's Complaint or Infringement Contentions. Nor are Defendants asserting any claim construction positions through these charts, including whether the preamble is a limitation. The portions of the prior art reference cited below are not exhaustive but are exemplary in nature.

Mastering Regular Expressions: Powerful Techniques for Perl and other Tools by Jeffrey E. F. Friedl, O'Reilly Press, 1st Edition, 7th Printing, ("Friedl") published in December, 1998 is prior art under at least 35 U.S.C. § 102(a) and 102(b), and 103(a). As described in the following claim chart, the asserted claims of U.S. Patent No. 6,842,796 (the "'796 Patent"), are invalid as anticipated by Friedl.

To the extent that Friedl is found not to anticipate one or more of the asserted claims of the '796 Patent, these claims are invalid as obvious in view of Friedl alone or in combination with other prior art references disclosed in Defendants' Invalidity Contentions and accompanying charts, including without limitation as set forth below.

'796 Patent	Friedl
[1P] A method of automatically processing an input sequence of data symbols, the method comprising the steps of:	To the extent the preamble is limiting, Friedl discloses this claim limitation explicitly, inherently, or as a matter of common sense, or it would have been obvious to add missing aspects of the limitation. Specifically, Friedl discloses methods of automatically processing text (i.e., input sequences of data symbols) via regular expressions. For example, see the following passages and/or figures, as well as all related disclosures:

EXHIBIT A – Friedl

'796 Patent	Friedl
	If you use a computer, you can benefit from regular expressions all the time (even if you don't realize it). When accessing World Wide Web search engines, with your editor, word processor, configuration scripts, and system tools, regular expressions are often provided as "power user" options. Languages such as Awk, Elisp, Expect, Perl, Python, and Tcl have regular-expression support built in (regular expressions are the very heart of many programs written in these languages), and regular-expression libraries are available for most other languages. For example, quite soon after Java became available, a regular-expression library was built and made freely available on the Web. Regular expressions are found in editors and programming environments such as *vi,* Delphi, Emacs, Brief, Visual C++, Nisus Writer, and many, many more. Regular expressions are very popular. Friedl at xv. One of ordinary skill would find this limitation disclosed either expressly or inherently in the teachings of this reference and its incorporated disclosures taken as a whole, or in combination with the state of the art at the time of the alleged invention. To the extent this reference is not found to teach this element explicitly, implicitly, or inherently, the element would have been obvious to one of ordinary skill in the art based on this reference, common sense, ordinary creativity of one of ordinary skill in the art, and the state of the art. Additionally, it would have been obvious to combine this reference with one or more other prior art references identified in Defendants' Invalidity Contentions Cover Pleading. Rather than repeat those disclosures here, they are incorporated by reference into this chart.
[1B] identifying at least a portion of information associated with the at least one regularly identifiable expression; and	Friedl discloses this claim limitation explicitly, inherently, or as a matter of common sense, or it would have been obvious to add missing aspects of the limitation. Specifically, Friedl discloses identifying portions of the information within input text based on the use of regular expression pattern matching (i.e., identifying at least a portion of information associated with the at least one regularly identifiable expression). For example, Friedl discloses identifying a double (repeated) word within a received text, or certain information which is associated with a regularly identifiable expression (e.g., text) in a mailbox file, such as the subjects, dates, or reply addresses that follow "Subject:", "Date:", and "Reply-To" prefixes, respectively, via the use of regular expressions.

EXHIBIT A – Friedl

’796 Patent	Friedl
	Further, insofar as it is a requirement of this limitation, Friedl discloses that the “regularly identifiable expression” and the “portion of information associated with the regularly identifiable expression” are not necessarily the same. *See Palo Alto Networks, Inc. v. Taasera Licensing LLC*, IPR No. 2023-443 (P.T.A.B. Aug. 15, 2023), Paper 7 at 9–10; *see, e.g.*, Friedl at 3–5, 7, 13–14, 25, 29, 48–56, 95–98, 112, 132–36. For example, as described above, Friedl identifies the regularly identifiable expression, “Subject:”; and Friedl identifies a portion of information associated with—but not the same as—“Subject:”; *i.e.*, Friedl identifies the text that follows “Subject:” *See* Friedl at 50 (“This [code snippet] attempts to match a string beginning with ‘Subject:*’. Once that much of the regex matches, the subsequent [.*] matches whatever else is on the rest of the line.”). For example, see the following passages and/or figures, as well as all related disclosures:

EXHIBIT A – Friedl

Double-word example in modern Perl

```
    $/ = ".\n";   1 # a special "chunk-mode"; chunks end with a period-newline
combination

    while (<>)    2
    {
        next unless s    3
         {#  (regex starts here)
                ###     Need to match one word:
                \b                  #  start of word....
                ( [a-z]+ )          #  grab word, filling $1(and \1).
                ### Now need to allow any number of spaces and/or <TAGS>
                (                   #  save what intervenes to $2.
                     (              #  ($3-parens onlyforgrouping the alternation)
                       \s           #  whitespace (includes newline, which is good).
                       |            #  -or-
                        <[^>]+>     #  item like < TAG>.
                     )+             #  need at least one of the above, but allow more.
                )

                ### Now match the first word again:
                (\1\b)              #  \b  ensures not embedded. This copy saved to $4.
            # (regex ends here)
          }
          # Above is the regex. Replacement string, below, followed by the modifiers, /i, /g,
and /x
          "\e[7m$1\e[m$2\e[7m$4\e[m"igx;     4

          s/^ ([^\e]*\n)+//mg;     5      #  Remove any unmarked lines.
          s/^/$ARGV: /mg;          6      #  Ensure lines begin with filename.
          print;
      }
```

value returned will be just one string, but a string that could potentially contain many of what we would consider to be logical lines.

EXHIBIT A – Friedl

'796 Patent	Friedl
	. . . 4 The replacement string is really just "`$1$2$4`" with a bunch of intervening ANSI escape sequences that provide highlighting to the two doubled words, but not to whatever separates them. These escape sequences are `\e[7m` to begin highlighting, and `\e[m` to end it. (`\e` is Perl's regex and string shorthand for the ASCII escape character, which begins these ANSI escape sequences.) Looking at how the parentheses in the regex are laid out, you'll realize that "`$1$2$4`" represents exactly what was matched in the first place. So other than adding in the escape sequences, this whole substitute command is essentially a (slow) no-op. We know that `$1` and `$4` represent matches of the same word (the whole point of the program!), so I could probably get by with using just one or the other in the replacement. However, since they might differ in capitalization, I use both variables explicitly. Friedl at 56, 57.

EXHIBIT A – Friedl

'796 Patent	Friedl

	To pull out the subject, we can employ a popular technique we'll use often:
	
	`if ($line =~ m/^Subject: (.*)/) {`
	`  $subject = $1;`
	`}`
	
	This attempts to match a string beginning with `'Subject:␣'`. Once that much of the regex matches, the subsequent `(.*)` matches whatever else is on the rest of the line. Since the `(.*)` is within parentheses, we can later use `$1` to access the text of the subject. In our case, we just save it to the variable `$subject`. Of course, if the regex doesn't match the string (as it won't with most), the result for the `if` is false and `$subject` isn't set for that line.
	
	Similarly, we can look for the `Date` and `Reply-To` fields:
	
	`if ($line =~ m/^Date: (.*)/) {`
	`  $date = $1;`
	`}`
	`if ($line =~ m/^Reply-To: (.*)/) {`
	`  $reply_address = $1;`
	`}`
	
	Friedl at 50.

EXHIBIT A – Friedl

'796 Patent	Friedl

	Page 96
	
	So, with a variable `$line` holding a string such as
	
	`Subject: Re: happy birthday`
	
	the Perl code
	
	`if ( $line =~ m/^Subject: (.*)/) {`
	`  print "The subject is: $1\n";`
	`}`
	
	produces `'The subject is: Re: happy birthday'`.
	
	To make the example more concrete, here's the snippet in Tcl
	
	`if [regexp "^Subject: (.*)" $line all exp1] {`
	`  puts "The subject is: $exp1"`
	`}`
	
	and in Python:
	
	`reg = regex.compile("Subject: \(.*\)")`
	`if reg.match(line) > 0:`
	`  print "The subject is:", reg.group(1)`
	
	As you can see, each language handles regular expressions in its own way, but the concept (and result) of greediness stays the same.
	
	Friedl at 96.

EXHIBIT A – Friedl

'796 Patent	Friedl

Removing the leading path from a filename

The ability to manipulate filenames is often useful. An example is removing a leading path from a full pathname, such as turning `/usr/local/bin/gcc` into `gcc`.

Stating problems in a way that makes solutions amenable is half of the battle. In this case, we want to remove anything up to (and including) the final slash. If there is no slash, it is already fine and nothing needs to be done.

Here, we really do want to use `.*`. With the regex `^.*/`, the `.*` consumes the whole line, but then backs off (that is, backtracks) to the last slash to achieve the match. Since a slash is our substitute command delimiter, we have to either escape it within the regex, as with **s/.*\///** (the regular expression is marked), which could make one dizzy, or (if supported) use a different delimiter, say **s!^.*/!!**.

If you have a variable `$filename`, the following snippets ensure that there is no leading path:

Language	Code Snippet
Perl	`$filename =~ s!^.*/!!;`
Tcl	`regsub "^.*/" $filename "" filename`
Python	`filename = regsub.sub("^.*/", "", filename)`

. . .

EXHIBIT A – Friedl

'796 Patent	Friedl

	Accessing the filename from a path
	
	A related option is to bypass the path and simply match the trailing filename part without the path, putting that text into another variable. The final filename is everything at the end that's not a slash: `[^/]*$`. This time, the anchor is not just an optimization; we really do need dollar at the end. We can now do something like:
	
	`$WholePath =~ m!([^/]*)$!;  # check variable $WholePath with regex.`
	`$FileName = $1;  # note text matched`
	
	You'll notice that I don't check to see whether the regex actually matches or not, because I *know* it will match every time. The only *requirement* of that expression is that the string have an end to match dollar, and even an empty string has an end.
	
	Thus, when I use `$1` to reference the text matched within the parenthetical subexpression, I'm assured it will have some (although possibly empty) value.*
	
	. . .

EXHIBIT A – Friedl

Both leading path and filename

The next logical step is to pick apart a full path into both its leading path and filename component. There are many ways to do this, depending on what we want. Initially, you might want to use `^(.*)/(.*)$` to fill `$1` and `$2` with the requisite parts. It looks like a nicely balanced regular expression, but knowing how greediness works, we are guaranteed that the first `.*` will never leave anything with a slash for `$2`. The only reason the first `.*` leaves anything at all is due to the backtracking done in trying to match the slash that follows. This leaves only that "backtracked" part for the later `.*`. Thus, `$1` will be the full leading path and `$2` the trailing filename.

One thing to note: we are relying on the initial `(.*)/` to ensure that the second `(.*)` does not capture any slash. We understand greediness, so this is okay. Still I like to be specific when I can, so I'd rather use `[^/]*` for the filename part. That gives us `^(.*)/([^/]*)$`. Since it shows exactly what we want, it acts as documentation as well.

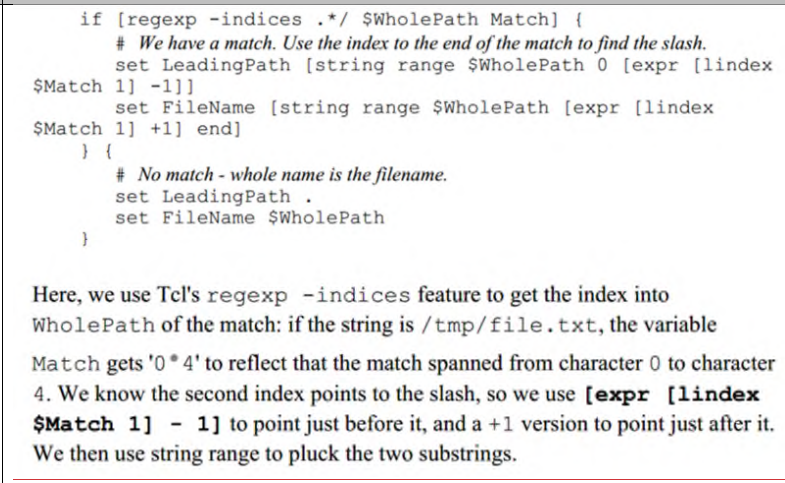
One big problem is that this regex requires at least one slash in the string, so if we try it on something like `file.txt`, there's no match, and no information. This can be a feature if we deal with it properly:

```
if ( $WholePath =~ m!^(.*)/(.*)$! ) {
    $LeadingPath = $1;
    $FileName = $2;
} else {
    $LeadingPath = ".";   # so "file.txt" looks like "./file.txt"
    $FileName = $WholePath;
}
```

Another method for getting at both components is to use either method for accessing the path or file, and then use the side effects of the match to construct the other. Consider the following Tcl snippet which finds the location of the last slash, then plucks the substrings from either side:

EXHIBIT A – Friedl

'796 Patent	Friedl

	
	`if [regexp -indices .*/ $WholePath Match] {`
	`  # We have a match. Use the index to the end of the match to find the slash.`
	`  set LeadingPath [string range $WholePath 0 [expr [lindex $Match 1] -1]]`
	`  set FileName [string range $WholePath [expr [lindex $Match 1] +1] end]`
	`} {`
	`  # No match - whole name is the filename.`
	`  set LeadingPath .`
	`  set FileName $WholePath`
	`}`
	
	Here, we use Tcl's `regexp -indices` feature to get the index into `WholePath` of the match: if the string is `/tmp/file.txt`, the variable `Match` gets '`0 4`' to reflect that the match spanned from character `0` to character `4`. We know the second index points to the slash, so we use **`[expr [lindex $Match 1] - 1]`** to point just before it, and a `+1` version to point just after it. We then use string range to pluck the two substrings.
	
	Friedl at 132–35.
	
	One of ordinary skill would find this limitation disclosed either expressly or inherently in the teachings of this reference and its incorporated disclosures taken as a whole, or in combination with the state of the art at the time of the alleged invention. To the extent this reference is not found to teach this element explicitly, implicitly, or inherently, the element would have been obvious to one of ordinary skill in the art based on this reference, common sense, ordinary creativity of one of ordinary skill in the art, and the state of the art. Additionally, it would have been obvious to combine this reference with one or more other prior art references identified in Defendants' Invalidity Contentions Cover Pleading. Rather than repeat those disclosures here, they are incorporated by reference into this chart.

[1C] extracting the portion of information.	Friedl discloses this claim limitation explicitly, inherently, or as a matter of common sense, or it would have been obvious to add missing aspects of the limitation.

Deleted: 34

EXHIBIT A – Friedl

'796 Patent	Friedl
	Contentions Cover Pleading. Rather than repeat those disclosures here, they are incorporated by reference into this chart.
[12B] (ii) identify at least a portion of information associated with the at least one regularly identifiable expression; and	Friedl discloses this claim limitation explicitly, inherently, or as a matter of common sense, or it would have been obvious to add missing aspects of the limitation. Specifically, Friedl discloses a computer and the computer programming languages Perl and Python for identifying portions of the information within input text based on the use of regular expression pattern matching (i.e., identifying at least a portion of information associated with the at least one regularly identifiable expression). For example, Friedl discloses identifying a double (repeated) word within a received text, or certain information which is associated with certain text in a mailbox file, such as the subjects, dates, or reply addresses that follow "Subject:", "Date:", and "Reply-To" prefixes, respectively, via the use of regular expressions. Further, insofar as it is a requirement of this limitation, Friedl discloses that the "regularly identifiable expression" and the "portion of information associated with the regularly identifiable expression" are not necessarily the same. *See Palo Alto Networks, Inc. v. Taasera Licensing LLC*, IPR No. 2023-443 (P.T.A.B. Aug. 15, 2023), Paper 7 at 9–10; *see, e.g.*, Friedl at 3–5, 7, 13–14, 25, 29, 48–56, 95–98, 112, 132–36. For example, as described above, Friedl identifies the regularly identifiable expression, "Subject:"; and Friedl identifies a portion of information associated with—but not the same as—"Subject:"; *i.e.*, Friedl identifies the text that follows "Subject:" *See* Friedl at 50 ("This [code snippet] attempts to match a string beginning with 'Subject:*'. Once that much of the regex matches, the subsequent [.*] matches whatever else is on the rest of the line."). For example, see the following passages and/or figures, as well as all related disclosures: *See above* re claim element [1B] which is incorporated by reference herein. One of ordinary skill would find this limitation disclosed either expressly or inherently in the teachings of this reference and its incorporated disclosures taken as a whole, or in combination with the state of the art at the time of the alleged invention. To the extent this reference is not found to teach this element explicitly, implicitly, or inherently, the element would have been obvious to one of ordinary skill in the art based on this reference, common sense, ordinary creativity of one of ordinary skill in the art, and the state of the art. Additionally, it would have been obvious to combine this reference with one or more other prior art references identified in Defendants' Invalidity